CLOUD-NATIVE APPLICATION MODERNIZATION IN MICROSERVICE ARCHITECTURE

***Hasna mol p*[1]**

ABSTRACT

Cloud computing has formed the conceptual and infrastructural basis for tomorrow's computing. The global computing infrastructure is rapidly moving towards Cloud-based architecture. While it is important to take advantage of Cloud-based computing by means of deploying it in diversified sectors,the main pillars of Cloud-Native applications are based on microservices architecture approaches, which can evolve with agility and scale to limits that would be difficult to achieve in a monolithic architecture deployed to either on-premises or Cloud environment. The key difference between a Cloud-native application and a simpler Cloud-Optimized web app is the recommendation to use microservice architectures in a Cloud-native approach [11]. Cloud-optimized apps can also be monolithic web apps or N-Tier apps. The microservice architecture is an advanced approach that onecan use for applications that are created from scratch, or when Cloud-native applicationsevolve from existing applications[6]. This paper presents a review on the Cloud-native application modernization in micro servicearchitecture [10]

Keyword : Microservices, cloud-native architecture, Software Modernization, cloud computing.

I. INTRODUCTION

Microservice architecture is an approach to building a server application as a set of small services. That means microservice architecture is mainly oriented to the backend, although the approach is also being used for the frontend. Each service runs its own process and communicates with other processes using protocols such as HTTP/HTTPS, Web Sockets, or AMQP [6]. Each microservice implements a specific end-to-end domain or business capability within a certain context boundary and each must be developed autonomously and is deployable independently [1].

Each microservice should own its related domain data model and domain logic (sovereignty and decentralized data management) based on different data storage technologies (SQL, No SQL) and different programming languages [7]. When developing a microservice, size should not be the important point. Instead, the important point should be to create loosely coupled [11] services so that we have autonomy of development, deployment and scale for each service [1]. Of course, when identifying and designing micro services, we should try to make them as small as possible, as long as you do not have too many direct dependencies with other microservices. More important than the size of the microservice is the internal cohesion it must have and its independence from other services [10].

Microservices enable better maintainability in complex, large and highlyscalable systems by letting

[1]Guest Lecturer, Department of Computer Application, Cochin University of Science and Technology.

you create applications based on many independently deployable services, of which eachhasgranular and autonomous lifecycles [4]. As an additional benefit, microservices can scale out independently. Instead of having a single monolithic application that you must scale out as a unit, you can scale out specific micro services[6]. That way, you can scale just the functional area that needs more processing power or network bandwidth to support demand, rather than scaling out other areas of the application that do not need to be scaled[7]. That means cost-saving, because you need less hardware.The difference between monolithic deployment approach and microservice application approach is shown diagrammatically in fig.1 [1].

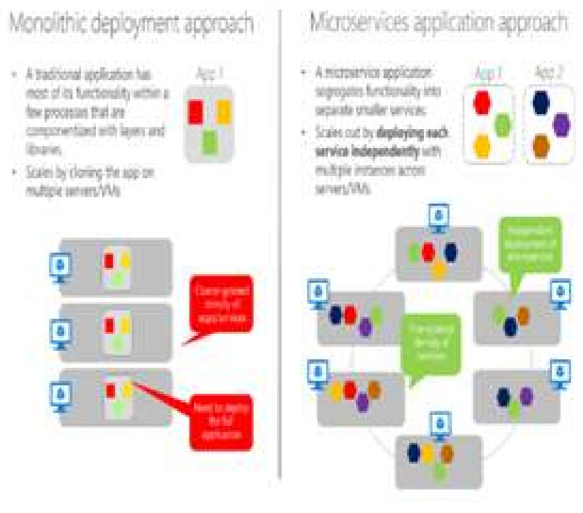
Monolithic deployment approach
Microservices application approach
App 1
App 1
App 2

Fig.1 The difference between monolithic deployment approach and microservice application approach [1]

The following are important aspects to enable success in going into production with a micro services-based system [1].

- Monitoring and health checks of the services and infrastructure.
- Scalable infrastructure for the services (that is, Cloud and orchestrators).
- Security design and implementation at multiple levels like authentication,authorization, secretsmanagement andsecure communication, etc.
- Rapid application delivery, usually with different teams focusing on different microservices.

1.1. Data sovereignty per microservice

An important rule for microservices architecture is that each microservice must own its domain data and logic. Just as a full application owns its logic and data, so must each microservice own its logic and data under an autonomous lifecycle, with independent deployment per microservice [6]. The following figure shows the data in traditional approach and microservice approach fig.2 [1].

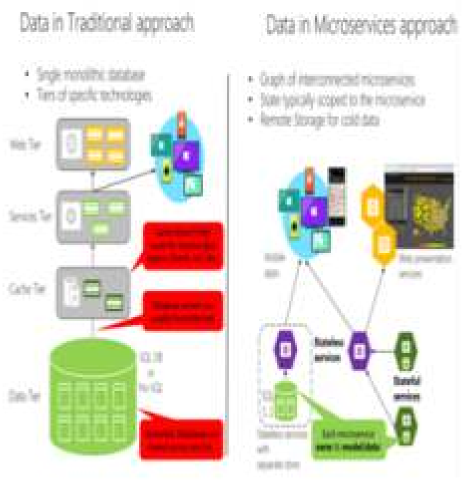

Fig.2 Data in traditional approach and data in microserviceapproach.

A monolithic application with typically a single relational database has two important benefits: ACID transactions and the SQL language, both working across all the tables and data related to your application. This approach provides a way to easily write a query that combines data from multiple tables [6]. Microservices-based applications often use a mixture of SQL and NoSQL databases, which aresometimes called the polyglot persistenceapproach [6].

2. Challenges and solutions for distributed data management

2.1 How to define the boundaries of each microservice

Defining microservice boundaries is probably the first challenge anyone encounters. Each microservice has to be apiece of your application and each microservice should be autonomous with all the benefits and challenges that it conveys. But how do you identify those boundaries? First, you need to focus on the application's logical domain models and related data [11].

2.2 How to create queries that retrieve data from several microservices

A second challenge is how to implement queries that retrieve data from several microservices, while avoiding chatty communication to the microservices from remote client applications [11].

2.2.1 API Gateway:For simple data aggregation from multiple microservices that own different databases, the recommended approach is an aggregation microservice referred to as an API Gateway [4]. However, we need to be careful about implementing this pattern, because it can be a choke point in your system, and it can violate the principle of micro serviceautonomy [6]. To mitigate this possibility, we can have multiple fined-grained API Gateways with each one focusing on a vertical "slice" or business area of the system [5].

2.2.2 CQRS (Command and Query Responsibility Segregation) with query/reads tables. Another solution for aggregating data from multiple microservices is the Materialized View pattern. In this approach, you generate, in advance (prepare denormalized data before the actual queries happen), a read-only table with the data that are owned by multiple micro services [4].

2.2.3 Cold data in central databases. For complex reports and queries that may not require real-time data, a common approach is to export your "hot data" (transactional data from the microservices) as "cold data" into large databases that are used only for reporting [5]. That central database system can be a Big Data-based system, like Hadoop, a data warehouse like one based on Azure SQL Data Warehouse, or even a single SQL database used just for reports (if size will not be an issue)[11].

2.3 How to achieve consistency across multiple microservices:

As stated previously, the data owned by each microservice is private to that microservice and can only be accessed using its microservice API. Therefore, a challenge presented is how to implement end-to-end business processes while keeping consistency across multiple microservices [3].

2.4 How to design communication across microservice boundaries

Communicating across microservice boundaries is a real challenge. In this context, communication does not refer to what protocol you should use (HTTP and REST, AMQP, messaging, and so on) failure and even occurrence oflarger outages. A popular approach is to implement HTTP (REST)[11] based microservices, due to their simplicity. An HTTP-based approach is perfectly acceptable; the issue here is related to how you use it. If you use HTTP requests and responses just to interact with your microservices from client applications or from API Gateways,that is fine. But if you create long chains of synchronous HTTP calls across microservices, communicating across their boundaries as if the microservices were objects in a monolithic application, your application will eventually run into problems [3].

- Blocking and low performance.
- Coupling microservices with HTTP
- Failure in any one microservice

Therefore, in order to enforce microservice autonomy and have better resiliency, the use of chains of request/response communication across microservices should be minimized [7]. For that only asynchronous interaction must be used for inter-microservice communication, either by asynchronous message- and event-based communication, or by (asynchronous) HTTP polling independently of the original HTTP request/response cycle [1].

3. Orchestrators in microservices

An "Orchestrator" is the general term for a piece of software that helps administratorsmanages these types of environments [11]. They are the components that take in requests like "I would like five copies of this service running in my environment." They try to make the environment match the desired state, no matter what happens.Orchestrators[6] (not humans) are what take action when a machine fails or a workload terminates for some unexpected reason. Most orchestrators do more than just deal with failure. Other features they have are managing new deployments, handling upgrades and dealing with resource consumption and governance [10]. All orchestrators are fundamentally about maintaining some desired state of configuration in the environment. Aurora on top of Mesos, Docker Datacenter/Docker Swarm, Kubernetes, and Service Fabric are the examples of orchestrators. These orchestrators are being actively developed to meet the needs of real workloads in production environments [2].

Conclusion

Building complex applications is inherently difficult. A Monolithic architecture only makes sense for simple, lightweight applications. We will end up in a world of pain, if we use it for complex applications. The Microservices architecture pattern is the better choice for complex, evolving applications despite the drawbacks and implementation challenges [1].

4. REFERENCES

[1]. Net Microservices architecture:Architecture for Containerized .Net Applications Cesar de la Torre, Bill Wagner, Mike Rousos .

[2]. Di Cosmo, R, Eiche, A, Mauro, J, Zacchiroli, S, Zavattaro, G and Zwolakowski, J (2015). Automatic Deployment of Services in the Cloud with Aeolus Blender. 13th International Conference on Service Oriented Computing, ICSOC.

[3]. Aderaldo, C.M, Mendonca, N.C, Pahl, C and Jamshidi, P (2017). Benchmark requirements for microservices architecture research. Proceedings of the 1st International Workshop on Establishing the CommunityWide Infrastructure for Architecture-Based Software Engineering.

[4]. Balalaie, A, Heydarnoori, A and Jamshidi, P (2016). Microservices Architecture Enables DevOps: Migration toa Cloud-Native Architecture. IEEE Software. Volume: 33, Issue: 3, May-June

[5]. Pahl, C, Brogi, A, Soldani, J and Jamshidi, P (2017). Cloud container technologies: a state-of-the-art review. IEEE Transactions on Cloud Computing.

[6]. Pahl, C, Jamshidi, P and Zimmermann, O (2018). Architectural principles for cloud software. ACM Transaction on Internet technology

[7]. B. Kitchen ham and P. Brereton. A systematic review of Systematic review process research in software engineering. Information and software technology, 55(12):2042075, 2013.

[8]. Z. Li, P. Liang, and P. Avgerinos. Application of Knowledge-based approaches in software architecture: Systematic mapping study. Information and Software Technology, 55(5):777-794, 2013

[9]. S. Newman. Building Micro services. O 'Reilly Median., 2015.

[10]. C. Pahl and P. Jamshidi. Micro services: A Systematic Mapping Study. In Proceedings of the 6th International Conference on Cloud Computing and Services Science Rome, Italy, pages 137-146, 2016

[11]. M. Richards. Micro services vs. Service-Oriented Architecture.O'Reilly Media, 2015.